Chapter 2. Assemblers

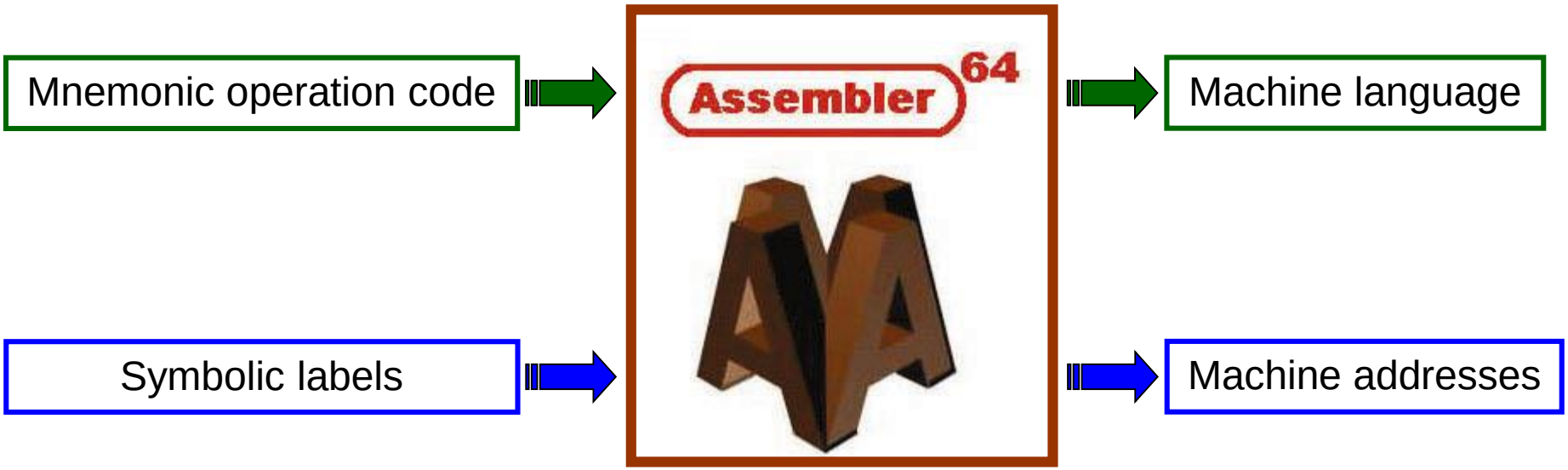
Outlines

- Fundamental functions of an assembler
  - A simple SIC assembler
  - Assembler algorithm and data structure
- Machine-dependent features
  - Instruction formats and addressing modes (SIC/XE)
  - Program relocation
- Machine-independent features
  - Literals
  - symbol-defining statements
  - Expressions
  - Program blocks
  - Control sections and program linking
- Design options: one-pass vs. multi-pass

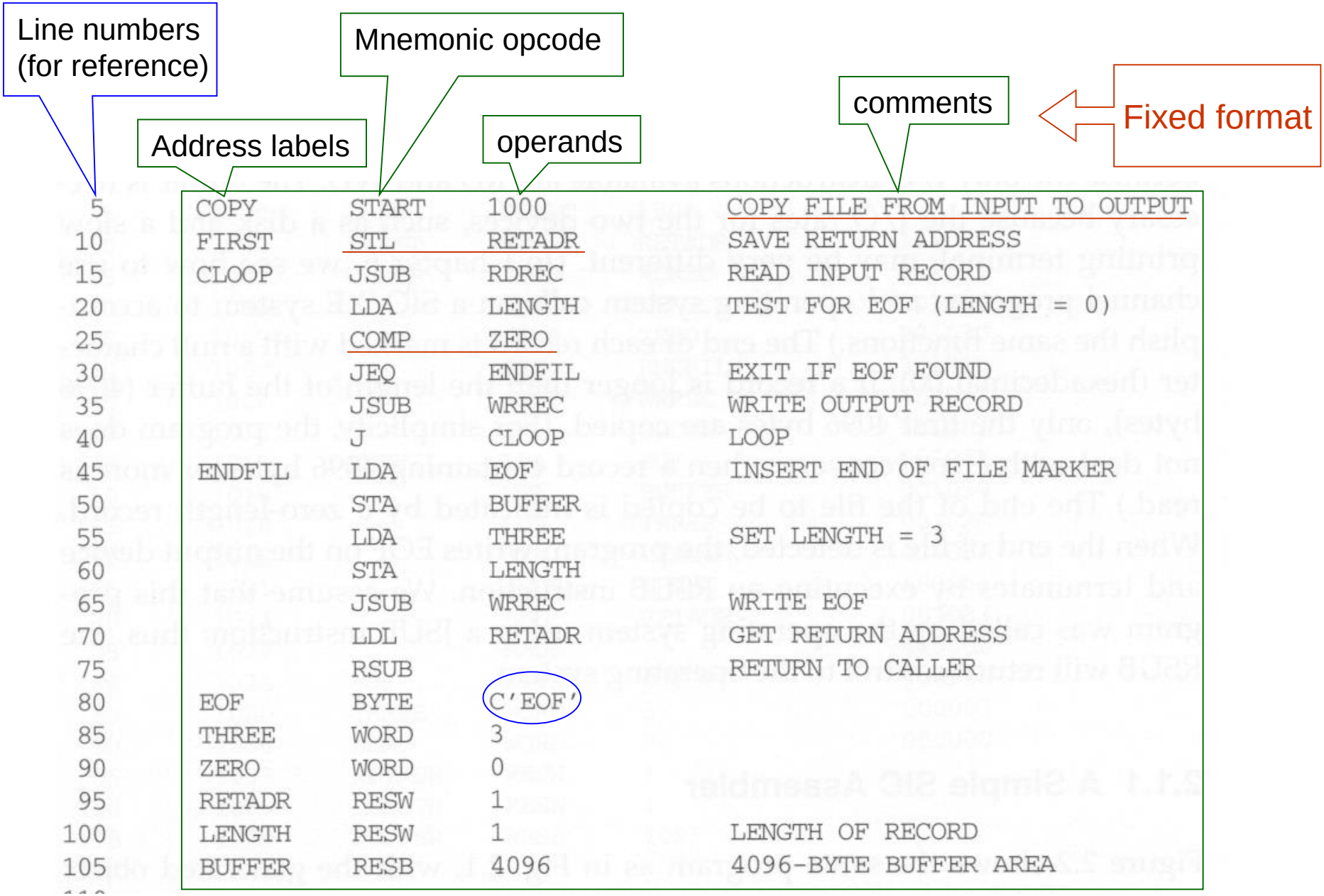
2.1 Basic SIC Assembler

Functions, Algorithm, and Data Structures

Fundamental Functions

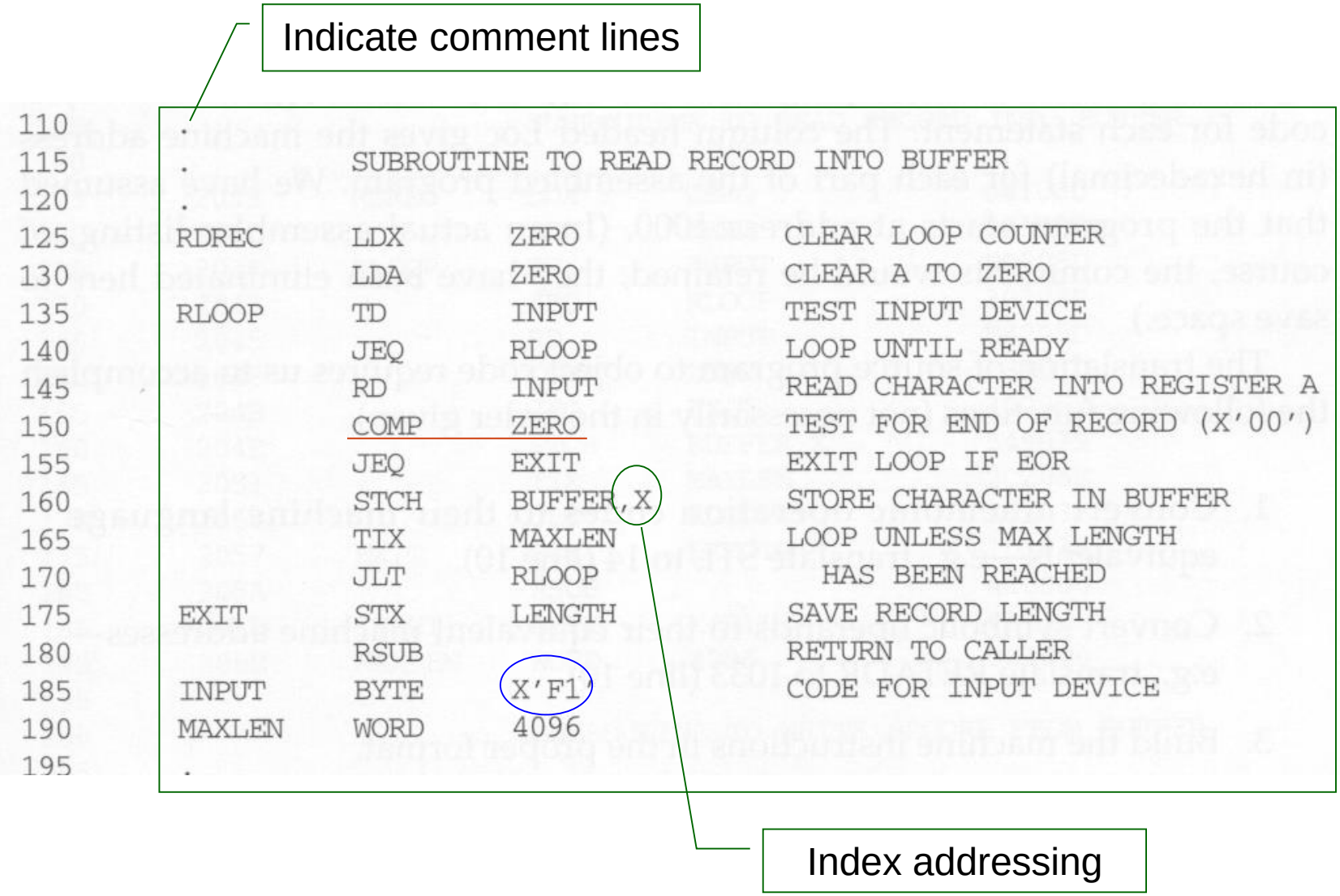


SIC Assembly Program




```
  5     COPY      START   1000        COPY FILE FROM INPUT TO OUTPUT
 10     FIRST     STL     RETADR      SAVE RETURN ADDRESS
 15     CLOOP     JSUB    RDREC       READ INPUT RECORD
 20               LDA     LENGTH      TEST FOR EOF (LENGTH = 0)
 25               COMP    ZERO
 30               JEQ     ENDFIL      EXIT IF EOF FOUND
 35               JSUB    WRREC       WRITE OUTPUT RECORD
 40               J       CLOOP       LOOP
 45     ENDFIL    LDA     EOF         INSERT END OF FILE MARKER
 50               STA     BUFFER
 55               LDA     THREE       SET LENGTH = 3
 60               STA     LENGTH
 65               JSUB    WRREC       WRITE EOF
 70               LDL     RETADR      GET RETURN ADDRESS
 75               RSUB                RETURN TO CALLER
 80     EOF       BYTE    C'EOF'
 85     THREE     WORD    3
 90     ZERO      WORD    0
 95     RETADR    RESW    1
100     LENGTH    RESW    1           LENGTH OF RECORD
105     BUFFER    RESB    4096        4096-BYTE BUFFER AREA
```

SIC Assembly Program



SIC Assembly Program

```
195  .
200  .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205  .
210  WRREC    LDX     ZERO        CLEAR LOOP COUNTER
215  WLOOP    TD      OUTPUT      TEST OUTPUT DEVICE
220           JEQ     WLOOP       LOOP UNTIL READY
225           LDCH    BUFFER,X    GET CHARACTER FROM BUFFER
230           WD      OUTPUT      WRITE CHARACTER
235           TIX     LENGTH      LOOP UNTIL ALL CHARACTERS
240           JLT     WLOOP         HAVE BEEN WRITTEN
245           RSUB                RETURN TO CALLER
250  OUTPUT   BYTE    X'05'       CODE FOR OUTPUT DEVICE
255           END     FIRST
```

Assembler Directives

- Basic assembler directives (pseudo instructions):
  - START :
    - Specify name and starting address for the program
  - END :
    - Indicate the end of the source program, and (optionally) the first executable instruction in the program.
  - BYTE :
    - Generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant.
  - WORD :
    - Generate one-word integer constant
  - RESB :
    - Reserve the indicated number of bytes for a data area
  - RESW :
    - Reserve the indicated number of words for a data area

SIC Assembler

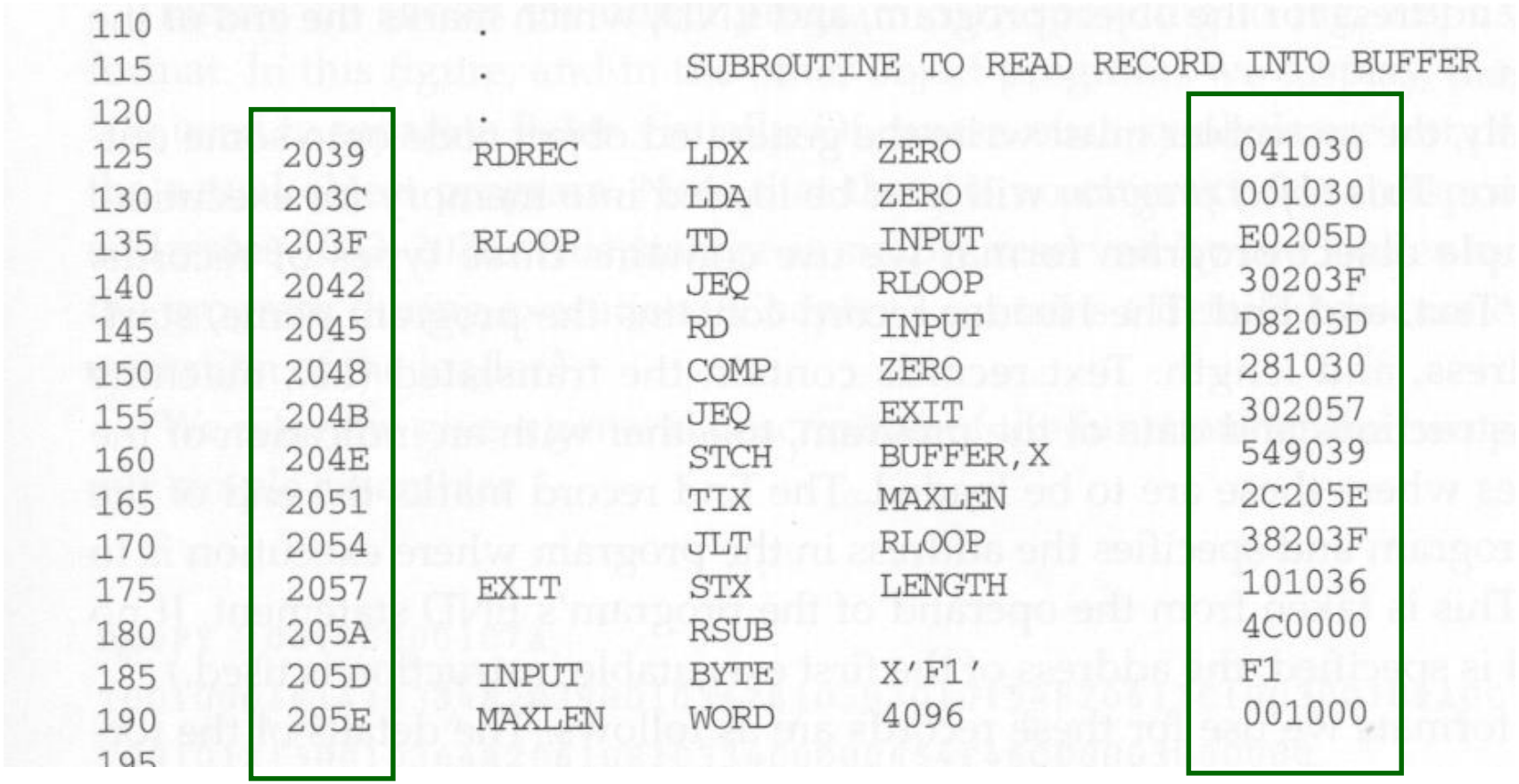
- Assembler's task:
  - Convert mnemonic operation codes to their machine language equivalents
  - Convert symbolic operands to their equivalent machine addresses (difficult)
  - Build machine instructions in proper format
  - Convert data constants into internal machine representations (data formats)
  - Write object program and the assembly listing

Assembly Program with Object Code

Line	Loc	Source statement			Object code
5	1000	COPY	START	1000	
10	1000	FIRST	STL	RETADR	141033
15	1003	CLOOP	JSUB	RDREC	482039
20	1006		LDA	LENGTH	001036
25	1009		COMP	ZERO	281030
30	100C		JEQ	ENDFIL	301015
35	100F		JSUB	WRREC	482061
40	1012		J	CLOOP	3C1003
45	1015	ENDFIL	LDA	EOF	00102A
50	1018		STA	BUFFER	0C1039
55	101B		LDA	THREE	00102D
60	101E		STA	LENGTH	0C1036
65	1021		JSUB	WRREC	482061
70	1024		LDL	RETADR	081033
75	1027		RSUB		4C0000
80	102A	EOF	BYTE	C'EOF'	454F46
85	102D	THREE	WORD	3	000003
90	1030	ZERO	WORD	0	000000
95	1033	RETADR	RESW	1	
100	1036	LENGTH	RESW	1	
105	1039	BUFFER	RESB	4096	
110		.			

Forward reference

Assembly Program with Object Code



```
110                 .
115                 .   SUBROUTINE TO READ RECORD INTO BUFFER
120                 .
125  2039   RDREC  LDX     ZERO        041030
130  203C          LDA     ZERO        001030
135  203F   RLOOP  TD      INPUT       E0205D
140  2042          JEQ     RLOOP       30203F
145  2045          RD      INPUT       D8205D
150  2048          COMP    ZERO        281030
155  204B          JEQ     EXIT        302057
160  204E          STCH    BUFFER,X    549039
165  2051          TIX     MAXLEN      2C205E
170  2054          JLT     RLOOP       38203F
175  2057   EXIT   STX     LENGTH      101036
180  205A          RSUB                4C0000
185  205D   INPUT  BYTE    X'F1'       F1
190  205E   MAXLEN WORD    4096        001000
195
```

Assembly Program with Object Code

```
195     .
200     .     SUBROUTINE TO WRITE RECORD FROM BUFFER
205     .
210  2061  WRREC  LDX     ZERO          041030
215  2064  WLOOP  TD      OUTPUT        E02079
220  2067         JEQ     WLOOP         302064
225  206A         LDCH    BUFFER,X      509039
230  206D         WD      OUTPUT        DC2079
235  2070         TIX     LENGTH        2C1036
240  2073         JLT     WLOOP         382064
245  2076         RSUB                  4C0000
250  2079  OUTPUT BYTE    X'05'         05
255                END     FIRST
```

Forward Reference

- Definition
  - A reference to a label that is defined later in the program
- Solution
  - Two passes
    - First pass: scan the source program for label definition, address accumulation, and address assignment
    - Second pass: perform most of the actual instruction translation

Forward Reference

- LOCCTR (Location Counter)
- SYMBOL TABLE

FIRST	1000
CLOOP	1003
BUFFER	1039
RDREC	2039

Forward Reference

- FIRST STL RETADR

  - - - - - - - - X - - - - - - - - - - - - - - - -

  0001 0100 0001 0000 0011 0011 ➔ 141033

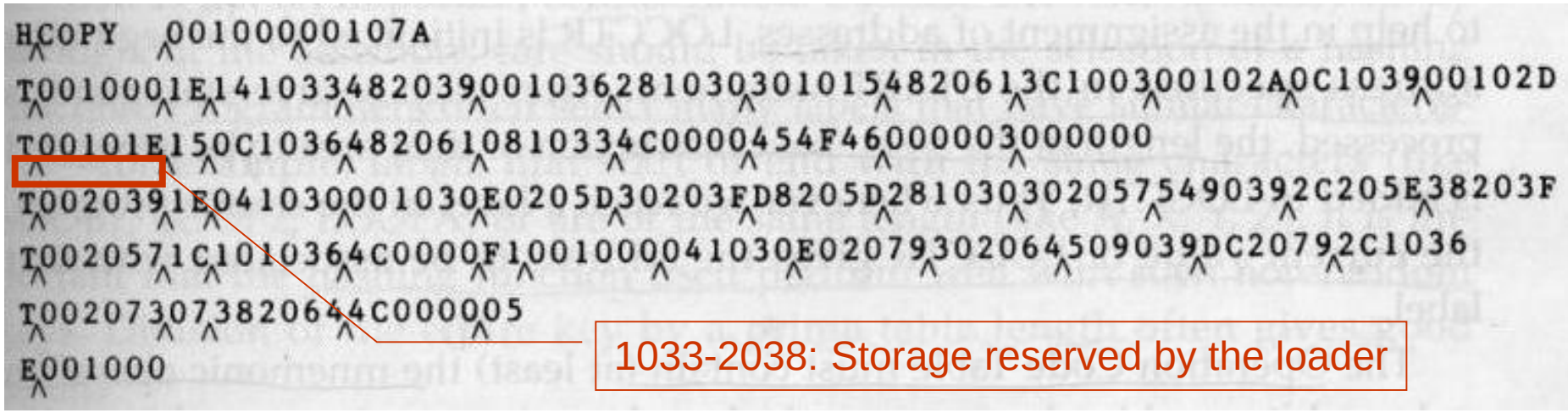
  STCH BUFFER,X

  - - - - - - - - X - - - - - - - - - - - - - - - -

  0101 0100 1001 0000 0011 1001 ➔ 549039

Object Program Format

- Header
  - Col. 1 — H
  - Col. 2~7 — Program name
  - Col. 8~13 — Starting address of object program (hex)
  - Col. 14-19 — Length of object program in bytes (hex)
- Text
  - Col.1 — T
  - Col.2~7 — Starting address for object code in this record (hex)
  - Col. 8~9 — Length of object code in this record in bytes (hex)
  - Col. 10~69 — Object code, represented in hex (2 col. per byte)
- End
  - Col.1 — E
  - Col.2~7 — Address of first executable instruction in object program (hex)

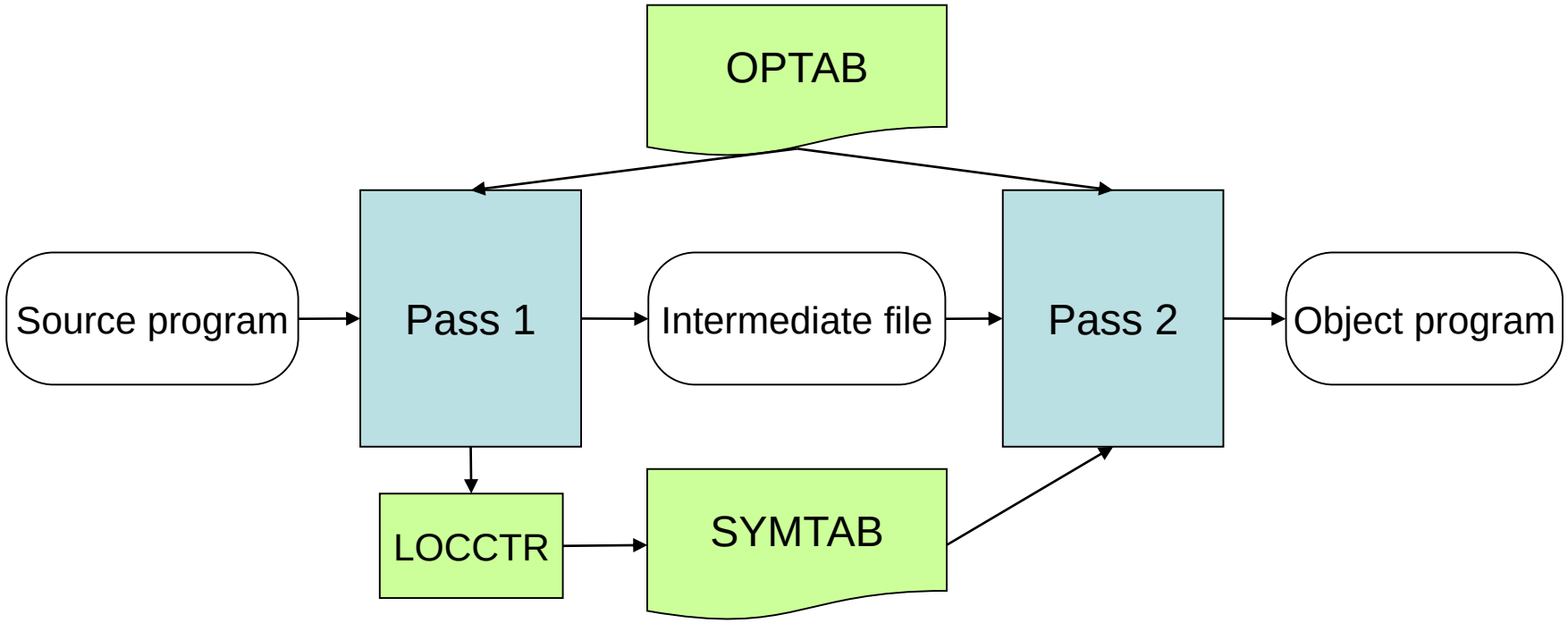


Two Pass SIC Assembler

- Pass 1 (define symbols)
  - Assign addresses to all statements in the program
  - Save the addresses assigned to all labels for use in Pass 2
  - Perform assembler directives, including those for address assignment, such as BYTE and RESW
- Pass 2 (assemble instructions and generate object program)
  - Assemble instructions (generate opcode and look up addresses)
  - Generate data values defined by BYTE, WORD
  - Perform processing of assembler directives not done during Pass 1
  - Write the object program and the assembly listing

Data Structures

- Operation Code Table (OPTAB)
- Symbol Table (SYMTAB)
- Location Counter (LOCCTR)



OPTAB

- Contents:
  - Mnemonic operation codes
  - Machine language equivalents
  - Instruction format and length
- During pass 1:
  - Validate operation codes
  - Find the instruction length to increase LOCCTR
- During pass 2:
  - Determine the instruction format
  - Translate the operation codes to their machine language equivalents
- Implementation: a static hash table

LOCCTR

- A variable accumulated for address assignment, i.e., LOCCTR gives the address of the associated label.
- LOCCTR is initialized to be the beginning address specified in the “start” statement.
- After each source statement is processed during pass 1, instruction length or data area is added to LOCCTR.

SYMTAB

- Contents:
  - Label name
  - Label address
  - Flags (to indicate error conditions)
  - Data type or length
- During pass 1:
  - Store label name and assigned address (from LOCCTR) in SYMTAB
- During pass 2:
  - Symbols used as operands are looked up in SYMTAB
- Implementation:
  - a dynamic hash table for efficient insertion and retrieval
  - Should perform well with non-random keys (LOOP1, LOOP2).

Pseudo Code for Pass 1

Pass 1:

```
begin
   read first input line
   if OPCODE = 'START' then
      begin
         save #[OPERAND] as starting address
         initialize LOCCTR to starting address
         write line to intermediate file
         read next input line
      end {if START}
   else
      initialize LOCCTR to 0
   while OPCODE ≠ 'END' do
      begin
         if this is not a comment line then
            begin
               if there is a symbol in the LABEL field then
                  begin
```

Pseudo Code for Pass 1

```
            search SYMTAB for LABEL
            if found then
                set error flag (duplicate symbol)
            else
                insert (LABEL,LOCCTR) into SYMTAB
        end {if symbol}
    search OPTAB for OPCODE
    if found then
        add 3 {instruction length} to LOCCTR
    else if OPCODE = 'WORD' then
        add 3 to LOCCTR
    else if OPCODE = 'RESW' then
        add 3 * #[OPERAND] to LOCCTR
    else if OPCODE = 'RESB' then
        add #[OPERAND] to LOCCTR
```

Pseudo Code for Pass 1

```
            else if OPCODE  = 'BYTE' then
                begin
                    find length of constant in bytes
                    add length to LOCCTR
                end {if BYTE}
            else
                set error flag (invalid operation code)
        end {if not a comment}
        write line to intermediate file
        read next input line
    end {while not END}
    write last line to intermediate file
    save (LOCCTR - starting address) as program length
end {Pass 1}
```

Pseudo Code for Pass 2

Pass 2:

```
begin
   read first input line {from intermediate file}
   if OPCODE = 'START' then
      begin
         write listing line
         read next input line
      end {if START}
   write Header record to object program
   initialize first Text record
   while OPCODE ≠ 'END' do
      begin
         if this is not a comment line then
            begin
               search OPTAB for OPCODE
```

Pseudo Code for Pass 2

```
if found then
   begin
      if there is a symbol in OPERAND field then
         begin
            search SYMTAB for OPERAND
            if found then
               store symbol value as operand address
            else
               begin
                  store 0 as operand address
                  set error flag (undefined symbol)
               end
         end {if symbol}
      else
         store 0 as operand address
      assemble the object code instruction
   end {if opcode found}
else if OPCODE = 'BYTE' or 'WORD' then
   convert constant to object code
```

Pseudo Code for Pass 2

```
            if object code will not fit into the current Text record then
               begin
                  write Text record to object program
                  initialize new Text record
               end
            add object code to Text record
         end {if not comment}
      write listing line
      read next input line
   end {while not END}
write last Text record to object program
write End record to object program
write last listing line
end {Pass 2}
```